Competitive Security Assessment

Zircuit-USDCAdapter

Sep 19th, 2024

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

• Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.

• Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.

• Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.

• Verify the code base is compliant with the most up-to-date industry standards and security best practices.

• Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

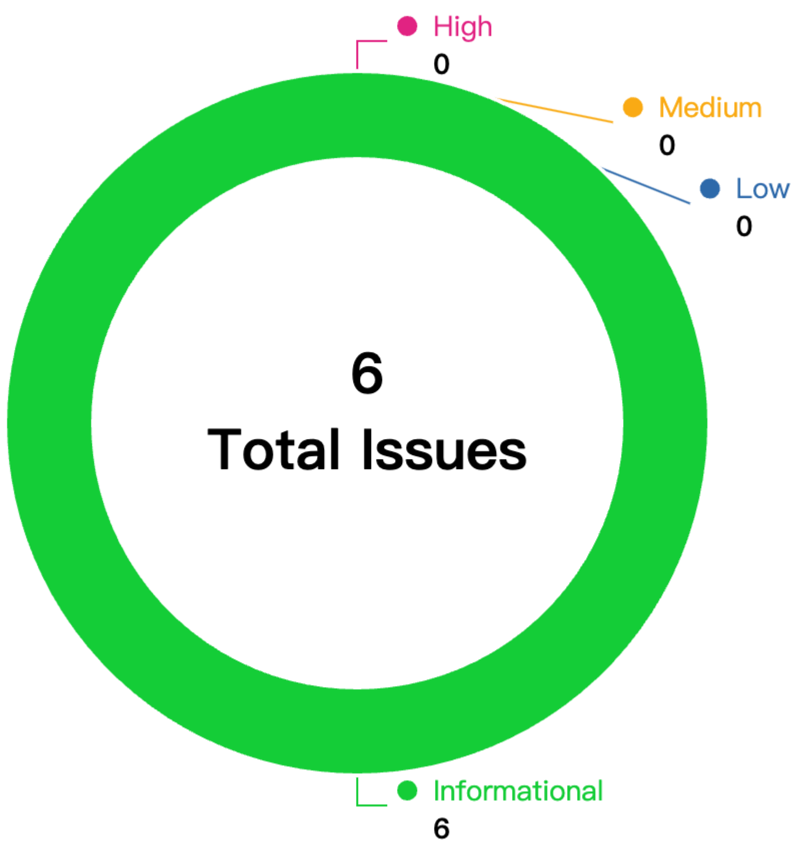
Overview

Project Name	Zircuit-USDCAdapter
Language	Solidity
Codebase	• https://github.com/zircuit-labs/USDCAdapter/tree/usdc_adapter • audit version-384a007dad38e4d7c1084ea41d6de7f0c43f7f4f • final version-41731c26c1994b89ea9b4c57a2d0b7534fc45b47

Audit Scope

File	SHA256 Hash
contracts/USDCAdapter.sol	98fd4751e1c78e32ab8d1966d8f3d252d3734e8db22e9d0bc9821491c68a1163

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
ZUA-1	Two-step ownership transfer	Privilege Rela ted	Informational	Fixed	***
ZUA-2	Missing keywords `override`	Code Style	Informational	Fixed	***
ZUA-3	Missing events	Code Style	Informational	Acknowledged	***
ZUA-4	Gas optimizing `supportInte rface()` function	Gas Optimiza tion	Informational	Fixed	***
ZUA-5	Critical addresses are set im mutable which can't be upda ted	Code Style	Informational	Acknowledged	***
ZUA-6	Code redundancy	Code Style	Informational	Fixed	***

ZUA-1:Two-step ownership transfer

Category	Severity	Client Response	Contributor
Privilege Related	Informational	Fixed	***

Code Reference

- code/contracts/USDCAdapter.sol#L44-L58

```
44: constructor(
45:         address _bridge,
46:         address _remoteToken,
47:         address _usdc,
48:         address _owner
49:     ) Ownable(_owner)
50:     {
51:         if (_bridge == address(0)) revert BridgeCannotBeZeroAddress();
52:         if (_remoteToken == address(0)) revert RemoteTokenCannotBeZeroAddress();
53:         if (_usdc == address(0)) revert USDCCannotBeZeroAddress();
54:
55:         BRIDGE = _bridge;
56:         REMOTE_TOKEN = _remoteToken;
57:         USDC = IUSDC(_usdc);
58:     }
```

Description

***: `Ownable2Step.sol` is safer than `Ownable.sol` for smart contracts because the owner cannot accidentally transfer smart contract ownership to a mistyped address. Rather than directly transferring to the new owner, the transfer only completes when the new owner accepts ownership.
Check the docs and the code here.

Recommendation

***: The OpenZeppelin's `Ownable2Step.sol` provides added safety due to its securely designed two-step process. Consider using `Ownable2Step.sol` instead of `Ownable.sol`.

Client Response

client response : Fixed. The issue has been fixed as recommended: zircuit-labs/USDCAdapter@41731c2

ZUA-2:Missing keywords `override`

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	***

Code Reference

- code/contracts/USDCAdapter.sol#L127
- code/contracts/USDCAdapter.sol#L134

```
127: function remoteToken() public view returns (address) {
```

```
134: function bridge() public view returns (address) {
```

Description

***: In Solidity, when a contract inherits from an interface and implements the functions declared in that interface, it's better to use the `override` keyword. This ensures clarity in the code and avoids ambiguity.

Recommendation

***: Adding `override` to these to functions:

```
    /**
    * @inheritdoc IOptimismMintableERC20
    */
    function remoteToken() public view override returns (address) {
        return REMOTE_TOKEN;
    }

    /**
    * @inheritdoc IOptimismMintableERC20
    */
    function bridge() public view override returns (address) {
        return BRIDGE;
    }
```

Client Response

client response : Fixed. The issue has been fixed as recommended: zircuit-labs/USDCAdapter@41731c2

ZUA-3:Missing events

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	***

Code Reference

- code/contracts/USDCAdapter.sol#L44-L58

```
constructor(
        address _bridge,
        address _remoteToken,
        address _usdc,
        address _owner
    ) Ownable(_owner)
    {
        if (_bridge == address(0)) revert BridgeCannotBeZeroAddress();
        if (_remoteToken == address(0)) revert RemoteTokenCannotBeZeroAddress();
        if (_usdc == address(0)) revert USDCCannotBeZeroAddress();

        BRIDGE = _bridge;
        REMOTE_TOKEN = _remoteToken;
        USDC = IUSDC(_usdc);
    }
```

Description

***: In the contract `USDCAdapter.sol` , `constructor` sets the `BRIDGE` , `REMOTE_TOKEN` , and `USDC` address, these are the key addresses for admin and user. It is a better practice to emit the corresponding events for transparency and readability.

Recommendation

***: Consider that emit corresponding events after setting key addresses.

Client Response

client response : Acknowledged. Events aren't necessary in this case since the address storage variables are only set once in the constructor and can be verified through public getter functions.

ZUA-4:Gas optimizing `supportInterface()` function

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	***

Code Reference

- code/contracts/USDCAdapter.sol#L112-L117

```
function supportsInterface(bytes4 _interfaceId) external pure virtual returns (bool) {
        bytes4 iface1 = type(IERC165).interfaceId;
        // Interface corresponding to the updated OptimismMintableERC20 (this contract).
        bytes4 iface2 = type(IOptimismMintableERC20).interfaceId;
        return _interfaceId == iface1 || _interfaceId == iface2;
    }
```

Description

***: The `supportsInterface()` function caches the supported interface to 2 arguments: `iface1` and `iface2`. Then check if the input matches those 2 arguments. Because this function will be used by the bridge, gas optimizing it will save gas when cross-chain transferring

Recommendation

***:

```
    function supportsInterface(bytes4 _interfaceId) external pure virtual returns (bool) {
-         bytes4 iface1 = type(IERC165).interfaceId;
-         bytes4 iface2 = type(IOptimismMintableERC20).interfaceId;
-         return _interfaceId == iface1 || _interfaceId == iface2;

+         return (_interfaceId == type(IERC165).interfaceId) || (_interfaceId == type(IOptimismMint
ableERC20).interfaceId);
    }
```

Client Response

client response : Fixed. The issue has been fixed as recommended: zircuit-labs/USDCAdapter@41731c2

ZUA-5:Critical addresses are set immutable which can't be updated

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	***

Code Reference

- code/contracts/USDCAdapter.sol#L26-L32

```
address public immutable REMOTE_TOKEN;

    /// @notice Address of the StandardBridge on this network.
    address public immutable BRIDGE;

    /// @notice Address of USDC contract (proxy) on this network.
    IUSDC public immutable USDC;
```

Description

***: `BRIDGE`, `REMOTE_TOKEN`, and `USDC` addresses are immutable, which leaves no room to update them if needed. The USDC contract is an upgradable contract and it can be upgraded in the future with a diff address which means its address is not fixed and if if `Remote token`/`Bridge` or `USDC` any of them changes their address in future there is no way to update them in the future and so adapter becomes unusable with the new implementation addresses.

```
address public immutable REMOTE_TOKEN;

    /// @notice Address of the StandardBridge on this network.
    address public immutable BRIDGE;

    /// @notice Address of USDC contract (proxy) on this network.
    IUSDC public immutable USDC;
```

Recommendation

***: Make these addresses updateable by the owner, with proper safeguards:

```
address public REMOTE_TOKEN;
address public BRIDGE;
IUSDC public USDC;

function updateCriticalAddresses(address newRemoteToken, address newBridge, address newUSDC) external onlyOwner {
    require(newRemoteToken != address(0) && newBridge != address(0) && newUSDC != address(0), "Invalid address");
    REMOTE_TOKEN = newRemoteToken;
    BRIDGE = newBridge;
    USDC = IUSDC(newUSDC);
    emit CriticalAddressesUpdated(newRemoteToken, newBridge, newUSDC);
}
```

Client Response

client response : Acknowledged. These stored addresses are intentionally made to be immutable to reduce centralization risk. If it's necessary to have different addresses, we can always deploy a new version of the USDC adapter.

ZUA-6:Code redundancy

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	***

Code Reference

- code/contracts/USDCAdapter.sol#L8

```
8: import { IERC20 } from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
```

Description

***: In the contract `USDCAdapter.sol`, it has imported `IERC20` but never be used. It will enlarge the bytecode size of the contract, which will result in more gas cost.

Recommendation

***: Recommend removing unused imports for gas saving.

Client Response

client response : Fixed. The issue has been fixed as recommended: zircuit-labs/USDCAdapter@41731c2

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.